Session 136

Sound and Networking for Games

David Hill and Todd Previte
Developer Technical Support

What You'll Learn

- Sound
  - API overview and options
  - Setup and initialization
  - Sound playback
  - MP3 playback

What You'll Learn

- Networking
  - BSD Sockets
    - Accessing
    - Setup and initialization
  - Network technologies suitable for games on Mac OS X
  - Tips and tricks for network gaming on Mac OS X

Technology Framework

- Sound
  - Carbon Sound Manager
  - QuickTime
  - Core Audio
- Networking
  - OpenTransport
  - Netsprocket/OpenPlay
  - BSD Sockets

Sound

Carbon Sound Manager

- Accessible from CFM or Mach-O
- Accessible from Carbon or Cocoa
- Supports variable bit-rate decoding

Sound

Carbon Sound Manager

- Several APIs are gone
  - SndPlayDoubleBuffer
    - DTS Sample: CarbonSndPlayDB
  - Disk-based APIs
  - Wavetable synth

Sound

Carbon Sound Manager

- Mac OS 8/9
  - Sound Manager was global
  - Hardware model provided by Sound Manager
- Mac OS X
  - Sound Manager is per-process
  - Hardware model provided by Core Audio

Carbon Sound Manager

- Setting Up a Channel

```
void soundCallback ( SndChannel* sc, SndCommand* cmd );

SndChannel*          channel;
ExtSoundHeader       header;

err = SndNewChannel( &channel, sampledSynth,
                     initStereo, soundCallback );
```

Carbon Sound Manager

- Filling Out a Sound Header

```
sound.samplePtr = soundDataPtr;
sound.numChannels = 2;
sound.sampleRate = rate44khz;
sound.encode = extSH;
sound.baseFrequency = 1;
sound.numFrames = kSamplesPerBuffer / 2;
sound.sampleSize = 16;
```

Carbon Sound Manager

- Sending Sound Commands

```
SndCommand soundCommand;

soundCommand.cmd = bufferCmd;
soundCommand.param1 = 0;
soundCommand.param2 = (int)&header;
SndDoCommand( channel, &soundCommand, true );

soundCommand.cmd = callbackCmd;
soundCommand.param1 = 0;
soundCommand.param2 = 0;
SndDoCommand( channel, &soundCommand, true );
```

Carbon Sound Manager

- Sending Sound Commands

```
SndCommand soundCommand;

soundCommand.cmd = bufferCmd;
soundCommand.param1 = 0;
soundCommand.param2 = (int)&header;
SndDoCommand( channel, &soundCommand, true );

soundCommand.cmd = callbackCmd;
soundCommand.param1 = 0;
soundCommand.param2 = 0;
SndDoCommand( channel, &soundCommand, true );
```

Carbon Sound Manager

- Sending Sound Commands

```
SndCommand soundCommand;

soundCommand.cmd = bufferCmd;
soundCommand.param1 = 0;
soundCommand.param2 = (int)&header;
SndDoCommand( channel, &soundCommand, true );

soundCommand.cmd = callbackCmd;
soundCommand.param1 = 0;
soundCommand.param2 = 0;
SndDoCommand( channel, &soundCommand, true );
```

Carbon Sound Manager

- Disposing of the Sound Channel

```
SndDisposeChannel( channel, true );
```

Sound

QuickTime

- MP3 and MIDI playback
- Still image loading
- Movie playing

QuickTime

- Playing an MP3

```
EnterMovies();
OpenMovieFile(&fsSpec, &refNum, fsRdPerm);
NewMovieFromFile(&movie, refNum, NULL,
    (StringPtr)NULL, newMovieActive, NULL);

// Play the movie once
StartMovie(movie);
while (!IsMovieDone(movie))
    MoviesTask(movie, 0);
DisposeMovie(movie);
CloseMovieFile(refNum);
ExitMovies();
```

QuickTime

- Playing an MP3

```
EnterMovies();
OpenMovieFile(&fsSpec, &refNum, fsRdPerm);
NewMovieFromFile(&movie, refNum, NULL,
    (StringPtr)NULL, newMovieActive, NULL);

// Play the movie once
StartMovie(movie);
while (!IsMovieDone(movie))
    MoviesTask(movie, 0);
DisposeMovie(movie);
CloseMovieFile(refNum);
ExitMovies();
```

QuickTime

- Playing an MP3

```
EnterMovies();
OpenMovieFile(&fsSpec, &refNum, fsRdPerm);
NewMovieFromFile(&movie, refNum, NULL,
    (StringPtr)NULL, newMovieActive, NULL);

// Play the movie once
StartMovie(movie);
while (!IsMovieDone(movie))
    MoviesTask(movie, 0);
DisposeMovie(movie);
CloseMovieFile(refNum);
ExitMovies();
```

QuickTime

- Playing an MP3

```
EnterMovies();
OpenMovieFile(&fsSpec, &refNum, fsRdPerm);
NewMovieFromFile(&movie, refNum, NULL,
    (StringPtr)NULL, newMovieActive, NULL);

// Play the movie once
StartMovie(movie);
while (!IsMovieDone(movie))
    MoviesTask(movie, 0);
DisposeMovie(movie);
CloseMovieFile(refNum);
ExitMovies();
```

QuickTime

- Playing an MP3

```
EnterMovies();
OpenMovieFile(&fsSpec, &refNum, fsRdPerm);
NewMovieFromFile(&movie, refNum, NULL,
    (StringPtr)NULL, newMovieActive, NULL);

// Play the movie once
StartMovie(movie);
while (!IsMovieDone(movie))
    MoviesTask(movie, 0);
DisposeMovie(movie);
CloseMovieFile(refNum);
ExitMovies();
```

QuickTime

- Playing an MP3

```
EnterMovies();
OpenMovieFile(&fsSpec, &refNum, fsRdPerm);
NewMovieFromFile(&movie, refNum, NULL,
    (StringPtr)NULL, newMovieActive, NULL);

// Play the movie once
StartMovie(movie);
while (!IsMovieDone(movie))
    MoviesTask(movie, 0);
DisposeMovie(movie);
CloseMovieFile(refNum);
ExitMovies();
```

QuickTime

- Playing an MP3

```
EnterMovies();
OpenMovieFile(&fsSpec, &refNum, fsRdPerm);
NewMovieFromFile(&movie, refNum, NULL,
    (StringPtr)NULL, newMovieActive, NULL);

// Play the movie once
StartMovie(movie);
while (!IsMovieDone(movie))
    MoviesTask(movie, 0);
DisposeMovie(movie);
CloseMovieFile(refNum);
ExitMovies();
```

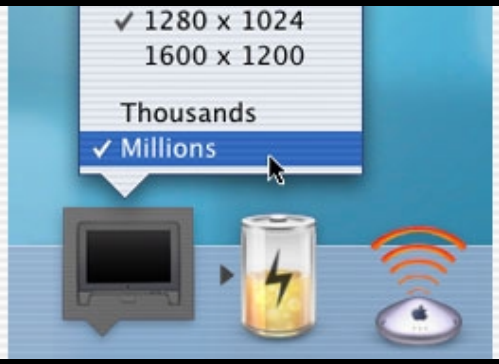
QuickTime

- Playing an MP3

```
EnterMovies();
OpenMovieFile(&fsSpec, &refNum, fsRdPerm);
NewMovieFromFile(&movie, refNum, NULL,
    (StringPtr)NULL, newMovieActive, NULL);

// Play the movie once
StartMovie(movie);
while (!IsMovieDone(movie))
    MoviesTask(movie, 0);
DisposeMovie(movie);
CloseMovieFile(refNum);
ExitMovies();
```

Demo



Sound

Core Audio

- Low-level Audio Interface
  - Audio device sharing
    - Processes can share an output device
  - Real-time audio interprocess communication
    - Processes can pass audio data around
  - Audio unit component model
    - String together multiple audio processing blocks

Sound

Core Audio

- Low Latency with Synchronized Output
- Uses Floating Point Samples [-1.0, 1.0]
  - Simplifies application code
  - Makes drivers work a bit harder
  - AltiVec (see vec_ctf) can really help here

Sound

Core Audio

- IOProcs
  - Time stamps
  - Input buffers
  - Output buffers
  - kAudioDevicePropertyStreamConfiguration

Sound

Core Audio

- Devices
  - One thread per device to call IOProcs
  - Mach real-time thread for timing accuracy
  - Device starts up with first IOProc
  - Device shuts down with last IOProc

Core Audio

- I/O Thread Pseudo-code (part 1)

```
while ( not done )
{
    wait until next processing time
    if ( device has input )
    {
        calculate input sample time
        copy data into IOProcs' input buffers
    }
```

Core Audio

- I/O Thread Pseudo-code (part 2)

```
if ( device has output )
{
    calculate output time
    Clear IOProcs' output buffers
}

call IOProcs
```

Core Audio

- I/O Thread Pseudo-code (part 3)

```
    if ( device has output )
    {
        mix IOProcs' output buffers
    }

}  //  End of while ( not done )
```

Sound

Core Audio

- Setup and Initialization
  - If Carbon CFM, load Core Audio via CFBundle
  - Gather device information
  - Set device properties
  - Add device I/O proc
  - Start the device

Core Audio Setup

- Loading Core Audio Via CFBundle

```
caBundleRef = CFBundleCreate(
    kCFAllocatorSystemDefault, bundleURL);

CFBundleLoadExecutable(caBundleRef );
funcPtrType caFuncPtr =
(funcPtrType)CFBundleGetFunctionPointerForName(
    caBundleRef , CFSTR(”caFuncName") );
```

Core Audio Setup

- Loading Core Audio Via CFBundle

```
caBundleRef = CFBundleCreate(
    kCFAllocatorSystemDefault, bundleURL);

CFBundleLoadExecutable(caBundleRef );
funcPtrType caFuncPtr =
(funcPtrType)CFBundleGetFunctionPointerForName(
    caBundleRef , CFSTR(”caFuncName") );
```

Core Audio Setup

- Loading Core Audio Via CFBundle

```
caBundleRef = CFBundleCreate(
    kCFAllocatorSystemDefault, bundleURL);

CFBundleLoadExecutable(caBundleRef );
funcPtrType caFuncPtr =
(funcPtrType)CFBundleGetFunctionPointerForName(
    caBundleRef , CFSTR(”caFuncName") );
```

Core Audio Setup

- Gathering Device Information

```
AudioDeviceID outputDeviceID;
UInt32 propertySize = sizeof(outputDeviceID);

status = AudioHardwareGetProperty (
    kAudioHardwarePropertyDefaultOutputDevice,
    &propertySize, &outputDeviceID);
```

Core Audio Setup

- Setting Device Properties

```
propertySize = sizeof(bufferByteCount);
bufferByteCount = kSamplesPerBuffer * sizeof(float);

status = AudioDeviceSetProperty(
            outputDeviceID, NULL, 0, NO,
            kAudioDevicePropertyBufferSize,
            propertySize, &bufferByteCount);
```

Core Audio Setup

- Adding the Device I/O Proc

```
status = AudioDeviceAddIOProc(
            outputDeviceID,
            audioDeviceIOProc, NULL);
```

Core Audio Setup

- Simple I/O Proc

```
OSStatus audioDeviceIOProc(AudioDeviceID inDevice,
                           const AudioTimeStamp *inNow,
                           const AudioBufferList *inInputData,
                           const AudioTimeStamp *inInputTime,
                           AudioBufferList *outOutputData,
                           const AudioTimeStamp *inOutputTime,
                           void *inClientData)
{
    //  For each sample frame
        //  For each channel
            //  Fill in the outOutputData buffer
}
```

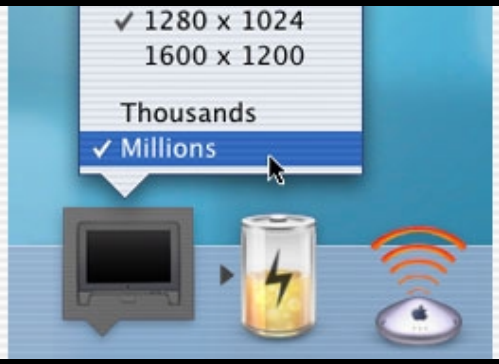
Core Audio Setup

- Starting the Device

```
status = AudioDeviceStart(outputDeviceID,
                          audioDeviceIOProc);

// Note that the I/O Proc may be called once
// before AudioDeviceStart returns!
```

Demo



Sound

API Comparison

- Raw Sound Playback
  - Core Audio, Sound Manager
- MP3 and MIDI Playback
  - QuickTime

Roadmap

209 Audio Processing and Sequencing Services	Room J2 **Fri., 3:30pm**
210 MIDI in Mac OS X	Room J2 **Fri., 5:00pm**
FF120 Feedback Forum: Games Technologies	Room J1 **Fri., 5:00pm**

Networking

Todd Previte
3D Graphics, Developer Technical Support

What You Will Learn

- BSD Sockets
  - Accessing
  - Setup and Initialization
- Network technologies suitable for games on Mac OS X
- Tips and tricks for network gaming on Mac OS X

Networking

- Mac OS X Networking
  - Carbon OT (Open Transport)
  - NetSprocket / OpenPlay
  - Sockets

Networking

- Mac OS X Open Transport
  - Easy application migration from Mac OS 8/9
  - Framework built upon sockets
    - OT API level compatibility only
  - Uses threads to emulate asynchronous mode
  - About 10% overhead over straight sockets
  - Only supports TCP/IP and DDP, ZIP and NBP for AppleTalk

OpenPlay/NetSprocket

- Overview
  - Open Source APIs
  - Cross platform
  - Carbonized for Mac OS X

OpenPlay/NetSprocket

- NetSprockets
  - High-level interface
  - Built on top of OpenPlay
  - Maintains most API compatibility
  - Use current Netsprocket documentation

OpenPlay/NetSprocket

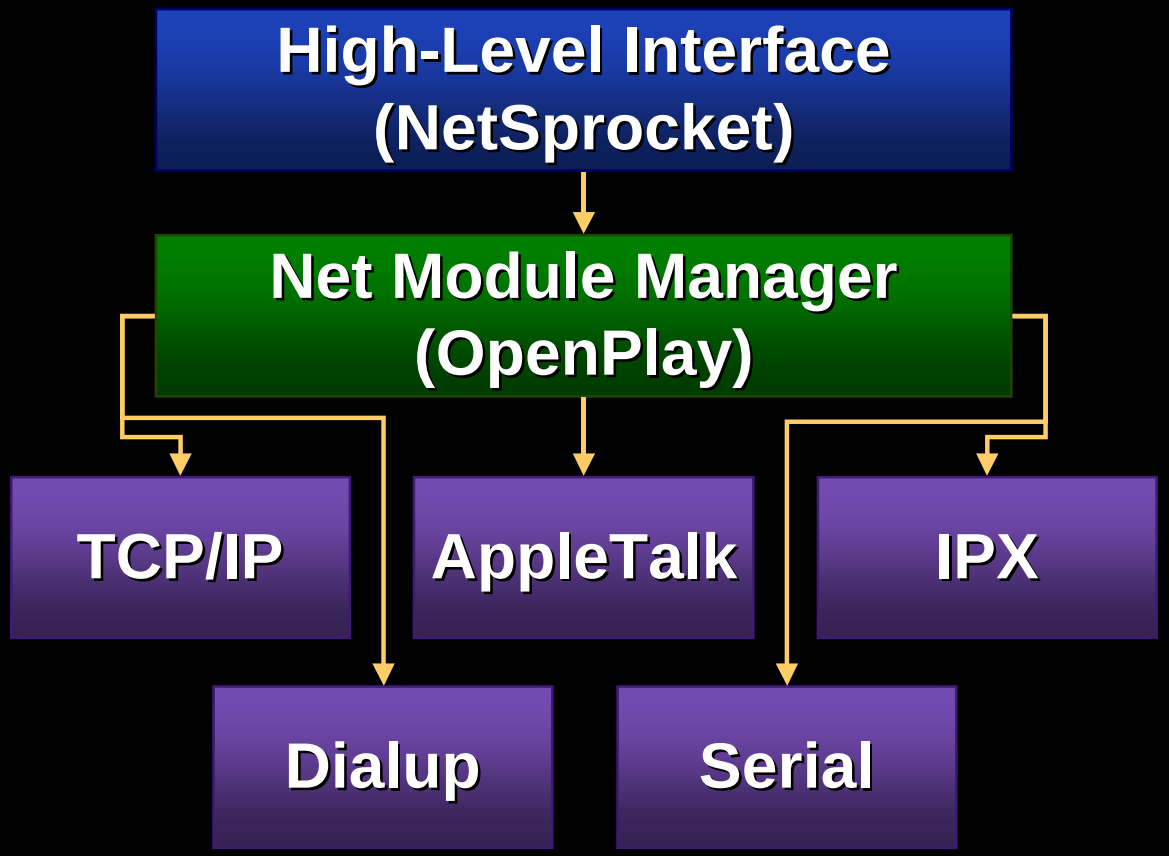
- OpenPlay
  - Low-level interface
  - “Network Module Manager”
    - Network Module = protocol module
    - Two ways for access
      - Directly through the OpenPlay layer
      - Through a high-level application interface

OpenPlay/NetSprocket

- OpenPlay
  - Provides 3 basic services
    - Configuration
    - Data transfer
    - Enumeration

OpenPlay/NetSprocket

- NetSprocket/OpenPlay Relationship



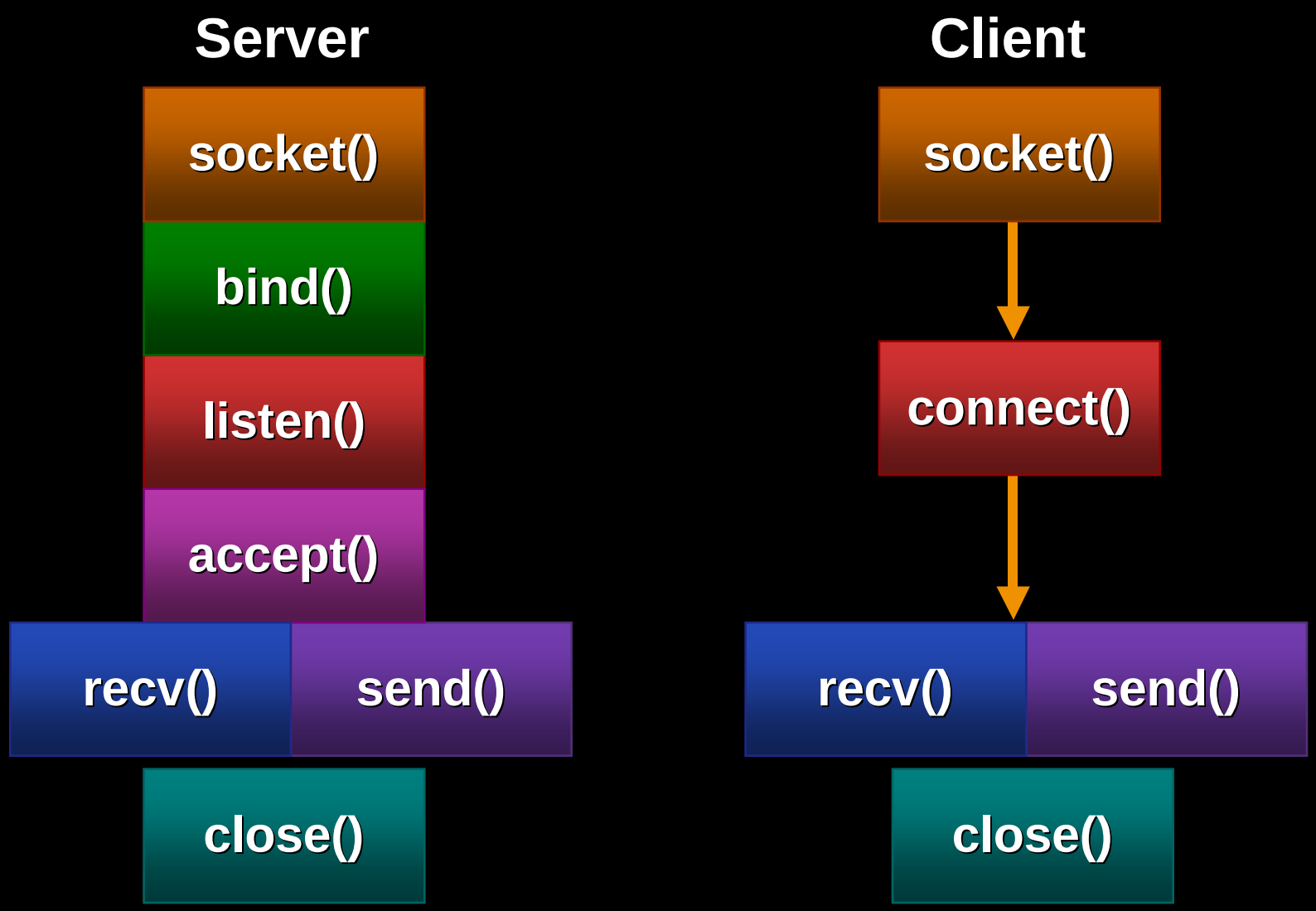
OpenPlay/NetSprocket

- NetSprocket/OpenPlay Information
  - Apple Open Source
    http://www.publicsource.apple.com/projects/openplay/
  - Includes sample code

BSD Sockets

- Overview
  - Standard UNIX Networking API
  - Sockets functions are synchronous

BSD Sockets


Server
socket()
bind()
listen()
accept()
recv()
send()
close()
Client
socket()
connect()
recv()
send()
close()

BSD Sockets

- Initialize Sockets Interface
  - CFM Carbon applications must load the System framework via CFBundle to access sockets
  - Mach-O Carbon applications must only include the System framework in the project
  - For loading frameworks with CFBundle see:

    CallMachOFramework

    http://developer.apple.com/samplecode/

BSD Sockets

- Initialize Sockets Interface
  - Cocoa applications have 2 options
    - NSSockets framework
    - Load the System framework and access directly

BSD Sockets

- Create Sockets(s)
  - One socket must be created per connection
  - socket() returns a socket descriptor
  - Socket descriptor is similar to file descriptor

BSD Sockets

- Create Sockets(s)

```
// Create a TCP socket
int sockFD = socket(AF_INET, SOCK_STREAM,
    IPPROTO_TCP);
// Create a UDP socket
int sockFD = socket(AF_INET, SOCK_DGRAM,
    IPPROTO_UDP);
// Create an ICMP socket
int sockFD = socket(AF_INET, SOCK_RAW,
    IPPROTO_ICMP);
```

BSD Sockets

- Establish Connection (Socket Association)
  - For a connection-oriented Server Socket:

```
struct sockaddr theAddr;
// Note that the sockaddr struct must be filled out
// properly
int addrLen = sizeof(theAddr);
int backlog = 2;

// Bind the socket to the local server address
int error = bind(sockFD, *theAddr, addrLen);

// Set the socket to listen for incoming connections
error =  listen(sockFD, backlog);
```

BSD Sockets

- Establish Connection (Socket Association)
  - Once an incoming connection has been detected
  - The accept() function returns a new socket descriptor

```
// Now we need to generate a new socket for this connection
int newSock =  accept(sockFD, *theAddr, addrLen);
```

BSD Sockets

- Establish Connection (Socket Association)
  - For a Connection-Oriented Client:
  - **sockfd** is the client socket on which to establish the connection
  - ***theAddr** is a pointer to an address to connect to
  - **addrlen** is the length of the address **theAddr**

```
struct sockaddr theAddr;
int addrlen = sizeof(theAddr);
int error = connect(sockfd, *theAddr, addrlen);
```

  - Connect() will assign a local address if bind() is not called (normally ok)

BSD Sockets

- Send and Receive Data
  - **send** writes data to a socket
  - **recv** reads data from a socket

```
int bytes, flags, error;
char buff[256];
// Send loop : bytes = # of bytes to send
// Send returns # of bytes sent if no error
error =  send(sockfd, *buff, bytes, flags);

// Receive loop : bytes = sizeof(recvBuffer)
error = recv(sockfd, *buff, bytes, flags);
```

BSD Sockets

- Shutdown and Cleanup
  - sockFD: Socket to be shutdown
  - Always read any pending data on the socket before closing
  - **how** specifies the method used to shutdown the socket
    - Offers more control over how a socket is disposed
  - Note that **close** is normally used, unless you need the extra control offered by **shutdown**

```
int how = 0;
int error = close(sockFD);
error = shutdown(sockFD, how);
```

Networking

- Tips and tricks for Mac OS X
  - OpenTransport applications
    - Port to sockets
    - Use the OT shim
  - Windows and UNIX applications
    - UNIX applications
      - Usually a straight-forward port
    - Winsock applications
      - Notable differences between Winsock and Sockets
    - DirectPlay applications

Networking

- OpenTransport
  - Using the OT shim
    - Easy path to Mac OS X from Mac OS 8/9
  - Porting to Sockets

Porting OT to Sockets

Opening and connecting

Endpoint provider OTOpenEndpoint()	Sockets file descriptor socket()
Active connection OTBind() -- qlen = 0 OTConnect()	bind() -- optional connect()
Passive connection OTBind() -- qlen > 0 OTListen() OTAccept()	bind() -- to specify port listen() -- backlog is qlen accept() -- return new file descriptor

Porting OT to Sockets

Sending and receiving

Sending data OTSnd() OTSndUData()	send() or sendmsg() sendto() or sendmsg()
Receiving data OTRcv() OTRcvUData()	recv() or recvmsg() recvfrom() or recvmsg()

Porting OT to Sockets

Disconnecting and closing

Abortive disconnect OTSndDisconnect()	Setsockopt(SO_LINGER, Abort)
Orderly disconnect OTSndOrderlyDisconnect() OTRcvOrderlyDisconnect()	shutdown(SHUT_WR) shutdown(SHUT_RD)
OtCloseProvider()	close()

Porting OT to Sockets

DNS

- Name to address resolution
  - gethostbyname()
- Address to name resolution
  - gethostbyaddr()
- Non-reentrant
- Blocking and not cancelable

Networking

- Windows Applications
  - DirectPlay
    - Platform-specific to Windows
    - Can be ported to NetSprocket
  - Winsock
    - Mostly maps onto sockets directly
    - Important differences

Networking

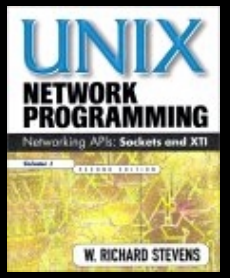
- Differences from Winsock
  - No asynchronous functions
    - Use threading to handle asynchronous operations
  - No pre-defined startup or shutdown routines; e.g., WSAStartup/WSACleanup
    - Not necessary on Mac OS X
  - WSA...() functions are Windows-specific
  - Windows "SOCKET" is not a 'socket'

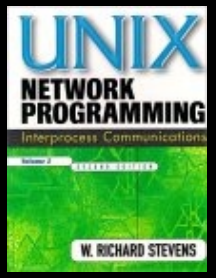
Summary

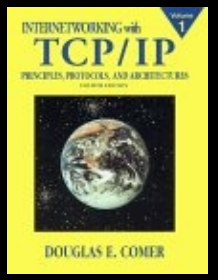
- Networking technologies on Mac OS X
  - OpenTransport
  - OpenPlay/NetSprocket
  - Sockets
- Setup and initialization of sockets
  - Carbon CFM
  - Carbon Mach-O
  - Cocoa
- Tips and Tricks
  - Using OT on Mac OS X
  - Porting OT to Sockets
  - Differences between Winsock and Sockets

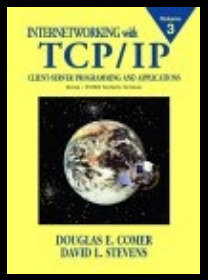
Reference

- Books
  - *Unix Network Programming*, Stevens
  - *Internetworking with TCP/IP*, Comer







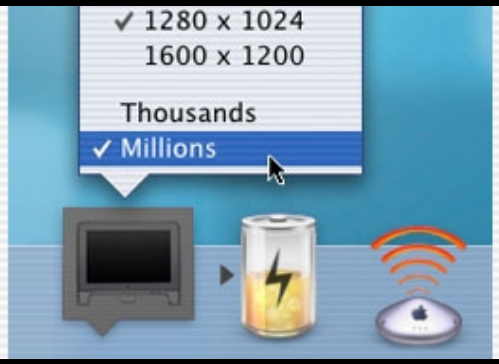


- lists.apple.com
  - OpenPlay developers' list—openplay_development
  - OpenTransport developers' list—opentransportdev
  - Darwin developers' list—darwin-development

Roadmap

300 Networking Overview	Room A1 **Tues., 9:00am**
304 Extensible Kernel Networking Services	Room J2 **Tues., 10:30am**
303 Network Configuration & Mobility	Room A1 **Tues., 5:00pm**
308 Network Services Location	Room C **Wed., 5:00pm**
310 AFP 3.0 & Appleshare Client on Mac OS X	Room J2 **Thur., 3:30pm**

Q&A



http://developer.apple.com/events/urls.html

WWDC
Worldwide Developers Conference 2001